

Instrucțiunile de control (structurile)

Începem acest capitol cu precizarea că în limbajul C++ toate structurile execută o singură instrucțiune. Dacă e necesar să fie executate mai multe instrucțiuni, se va utiliza instrucțiunea compusă (mai multe instrucțiuni cuprinse între paranteze accolade sunt interpretate de compilator ca și cum ar fi o singură instrucțiune, numită instrucțiunea compusă).

Structura alternativă (de decizie)

Structura alternativă are rolul de a alege o alternativă (care constă în execuția o singură dată a unei instrucțiuni) dintre mai multe alternative (două sau mai multe instrucțiuni care se pot executa, dar numai una dintre ele este aleasă pentru execuție).

Structura alternativă simplă. Instrucțiunea **if...else**

Sintaxa:

```
if (expresie)
    instrucțiune_1;
[else
    instrucțiune_2;]
```

Instrucțiunea se execută astfel:

Se evaluează **expresie** și dacă rezultatul e diferit de zero (corespunzător valorii logice *True*), se execută **instrucțiune_1**, altfel, se execută **instrucțiune_2**.

Observații:

1. Partea **else instrucțiune_2**; poate să lipsească. În acest caz doar una dintre alternative se execută.
2. Fiecare alternativă execută o singură instrucțiune. Dacă e necesară execuția mai multor instrucțiuni, atunci ele vor fi grupate între paranteze accolade (fiind considerate o singură instrucțiune, instrucțiunea compusă).
3. **instrucțiune** poate fi orice instrucțiune recunoscută de C++, inclusiv instrucțiunea **if**. Când două sau mai multe instrucțiuni **if** sunt incluse una în alta, se spune că avem instrucțiuni **if imbricate**.

Structura alternativă generalizată. Instrucțiunea **switch...case**

Sintaxa:

```
switch (expresie)
{
    case exp_1: instrucțiune_1 break;
    case exp_2: instrucțiune_2 break;
    .....
    case exp_i: instrucțiune_i break;
    .....
    case exp_n: instrucțiune_n break;
    [default:   instrucțiune_n+1]
}
```

Instrucțiunea se execută astfel:

Se evaluează **expresie** și apoi:

dacă rezultatul obținut este egal cu valoarea obținută în urma evaluării:

- **exp_1**, se execută **instrucțiune_1**, după care se trece la executarea instrucțiunii care urmează după **switch**;
- **exp_2**, se execută **instrucțiune_2**, după care se trece la executarea instrucțiunii care urmează după **switch**;
- **exp_2**, se execută **instrucțiune_2**, după care se trece la executarea instrucțiunii care urmează după **switch**;
-
- **exp_i**, se execută **instrucțiune_i**, după care se trece la executarea instrucțiunii care urmează după **switch**;
- **exp_n**, se execută **instrucțiune_n**, după care se trece la executarea instrucțiunii care urmează după **switch**;

altfel se execută instrucțiunea **instrucțiune_n+1** după care se trece la executarea instrucțiunii care urmează după **switch**.

Observații:

1. **expresie** trebuie să furnizeze un rezultat numeric întreg;

2. Rezultatele pentru fiecare caz în parte al **exp_i** trebuie să fie valori constante întregi;
3. Toate expresiile corespunzătoare cazurilor (**exp_i**) trebuie să fie diferite între ele;
4. Caracterul **:** este obligatoriu, fiind un separator între eticheta cazului respectiv (**case exp_i**) și instrucțiunea care se va executa (**instrucțiune_i**);
5. Colecția de instrucțiuni etichetate trebuie să fie încapsulată într-un bloc delimitat de accolade;
6. Eticheta default este opțională. Instrucțiunea atașată **instrucțiune_n+1** se execută numai dacă nu a fost îndeplinit nici un caz anterior;
7. Instrucțiunea **break**; este obligatorie la fiecare caz în parte și are ca efect întreruperea execuției instrucțiunii **switch...case** și trecerea la executarea următoarei instrucțiuni din program. Dacă ea lipsește, atunci e verificat și următorul caz.

Structura repetitivă

Structura repetitive are rolul de a repeta execuția unei instrucțiuni sau a unui grup de instrucțiuni. Aceste structuri au fost introduse în limbajele de programare cu scopul de a ușura munca programatorului ca să nu repete scrierea în mod repetat a aceleiași instrucțiuni sau a unui grup de instrucțiuni în programul sursă. De cele mai multe ori, nici nu e posibil să implementăm un algoritm scriind în mod repetat aceste instrucțiuni pentru că în majoritatea cazurilor nu știm de câte ori ele trebuie repetate.

Structura repetitivă condiționată anterior

Instrucțiunea **while**

Sintaxa:

```
while (expresie)
    instrucțiune;
```

Instrucțiunea se execută astfel:

Se evaluează **expresie** și dacă rezultatul este diferit de zero (corespunzător valorii logice *True*), se execută **instrucțiune**, apoi se evaluează din nou **expresie** și dacă rezultatul este diferit de zero se execută din nou **instrucțiune** și așa mai departe (deci, se repetă execuția instrucțiunii **instrucțiune** atâta timp cât evaluarea expresiei **expresie** furnizează un rezultat diferit de zero). În momentul în care evaluarea expresiei este zero (corespunzător valorii *False*), se trece la execuția instrucțiunii care urmează instrucțiunii **while**.

Observații:

1. Structura **while** repetă o singură instrucțiune. Dacă este necesară repetarea unui grup de instrucțiuni, atunci ele vor fi încapsulate într-o instrucțiune compusă.
2. **while** este o structură repetitivă cu un număr necunoscut de pași condiționată anterior.
3. Instrucțiunea care se repetă poate fi chiar și o instrucțiune vidă.
4. **expresie** poate fi formată și dintr-o expresie compusă din mai multe expresii legate cu operatorul virgulă (vezi exemplele de mai jos).
5. E obligatoriu ca instrucțiune să modifice valorile unor anumite variabile care intervin în expresie, astfel încât valoarea acesteia (după un număr finit de pași) să fie zero, făcând posibilă ieșirea din ciclu, altfel am avea un **ciclu infinit**.
6. Deoarece mai întâi se evaluează **expresie** și numai dacă valoarea sa e diferită de zero, se execută **instrucțiune**, spunem că structura **while** este o structură repetitivă **condiționată anterior** (evaluarea expresiei se face anterior execuției instrucțiunii).
7. Întrucât evaluarea expresiei se face anterior execuției instrucțiunii, dacă rezultatul evaluării este zero chiar de la început, atunci **instrucțiune** nu se va executa deloc.
8. **instrucțiune** poate fi orice instrucțiune recunoscută de C++, inclusiv instrucțiunea **while**. Când două sau mai multe instrucțiuni **while** sunt incluse una în alta, se spune că avem instrucțiuni **while** **imbricate**.

Exemple:

```
#include <iostream>
using namespace std;
int main()
{
    // Se citesc numere naturale pana apare 0 si se afiseaza suma celor pare
    unsigned long long int a,s=0;
    cin>>a;
    while(a)    //Expresia a condenseaza expresia a!=0
    {
        if(!(a%2)) // Expresia !(a%2) condenseaza expresia a%2==0
            s+=a;    //Expresia s+=a condenseaza expresia s=s+a
        cin>>a;
    }
    cout<<s;
    return 0;
}
```

```
#include <iostream>
using namespace std;
int main()
{
    // Se citeste un numar natural si se afiseaza suma cifrelor
    unsigned long long int n;
    unsigned s=0;
    cin>>n;
    while(s+=n%10,n/=10)    // Expresie compusa
        ;                  // Se repeta instructiunea vida
    cout<<s;
    return 0;
}
```

Expresia compusă cu ajutorul operatorului virgulă din acest exemplu se evaluează astfel: mai întâi se evaluează prima expresie (cu care se actualizează suma cifrelor s), apoi se evaluează a doua expresie cu care se elimină ultima cifră din număr și rezultatul acestei expresii este memorat în variabila n (expresia fiind o atribuire). Cum valoarea expresiei compuse din mai multe expresii legate cu operatorul virgule este dat de ultima expresie, aici valoarea expresiei compuse este chiar valoarea lui n.

Instrucțiunea for

Sintaxa:

```
for (expresie_1;expresie_2;expresie_3)
    instrucțiune;
```

Instrucțiunea se execută astfel:

Mai întâi se evaluează **expresie_1** o singură dată înainte de prima execuție a instrucțiunii și are rolul de **inițializare**. Apoi se evaluează **expresie_2** având rolul de **testare**, iar dacă rezultatul evaluării este diferit de zero (corespunzător valorii *True*) se va executa **instrucțiune**. După aceea se evaluează **expresie_3** care are rolul de **modificare** prin schimbarea stării curente, astfel încât să se avanseze spre starea finală.

În continuare se evaluează **expresie_2** și dacă valoarea ei e diferită de zero se execută din nou instrucțiune, după care se evaluează **expresie_3** pentru a modifica starea curentă și tot așa până când, în urma evaluării expresiei **expresie_2** se obține valoarea zero (corespunzător valorii *False*), moment în care se încheie execuția structurii **for** și se trece la execuția instrucțiunii care urmează după ea.

Observații:

1. Oricare dintre cele trei expresii pot să lipsească (pot să lipsească chiar și toate), dar cei doi delimitatori ; dintre cele trei expresii sunt obligatorii. Când o expresie lipsește putem considera că acolo figurează o expresie vidă. Structura for(;;) are toate expresiile vide și ea execută un **ciclu infinit**. Un program care conține un asemenea ciclu se numește **program care ciclează la infinit**. E de la sine înțeles că dacă lipsește **expresie_2** instrucțiunea va executa un ciclu infinit.
2. Instrucțiunea **for** este tot o instrucțiune condiționată anterior ca și **while**, deoarece mai întâi se evaluează expresia (în cazul acesta **expresie_2**) și abia după aceea se execută sau nu **instrucțiune**.
3. În unele limbaje de programare instrucțiunea for este considerată o structură repetitivă cu un număr cunoscut de pași, dar limbajul C++ e mult mai flexibil și ea poate fi utilizată și în situații în care numărul de pași e necunoscut.
4. Deoarece mai întâi se evaluează **expresie_2** și numai dacă valoarea sa e diferită de zero, se execută **instrucțiune**, spunem că structura **for** este o structură repetitivă **condiționată anterior** (evaluarea expresiei se face anterior execuției instrucțiunii).
5. Întrucât evaluarea expresiei se face anterior execuției instrucțiunii, dacă rezultatul evaluării este zero chiar de la început, atunci **instrucțiune** nu se va executa deloc.
6. **instrucțiune** poate fi orice instrucțiune recunoscută de C++, inclusiv instrucțiunea **for**. Când două sau mai multe instrucțiuni **for** sunt incluse una în alta, se spune că avem instrucțiuni **for imbricate**.

Exemple:

Structura repetitivă condiționată posterior Instrucțiunea do...while

Sintaxa:

```
do
    instrucțiune;
while (expresie);
```

Instrucțiunea se execută astfel:

Se execută instrucțiune după care se evaluează **expresie**. Dacă rezultatul evaluării expresiei este diferit de zero (corespunzător valorii *True*), se execută din nou instrucțiune, apoi se evaluează din nou expresie și tot așa până

când valoarea expresiei va fi zero (corespunzător valorii *False*), se trece la execuția instrucțiunii care urmează instrucțiunii **do...while**.

Observații:

1. Structura **do...while** repetă o singură instrucțiune. Dacă este necesară repetarea unui grup de instrucțiuni, atunci ele vor fi încapsulate într-o instrucțiune compusă.
2. Spre deosebire de **while**, structura **do..while** este o **structură repetitivă cu un număr necunoscut de pași condiționată posterior**. Din acest motiv, **instrucțiune** se execută cel puțin odată, chiar dacă **expresie** are de la început valoarea zero. Prin urmare, când utilizăm structura **do...while** trebuie să fim foarte atenți deoarece putem să avem erori logice când implementăm un algoritm și utilizăm această structură.
3. Instrucțiunea care se repetă poate să lipsească (se consideră că există acolo o instrucțiune vidă), dar **expresie** nu poate să lipsească niciodată (absența ei generează o eroare de sintaxă).
4. **expresie** poate fi formată și dintr-o expresie compusă din mai multe expresii legate cu operatorul virgulă (vezi exemplele de mai jos).
5. E obligatoriu ca instrucțiune să modifice valorile unor anumite variabile care intervin în expresie, astfel încât valoarea acesteia (după un număr finit de pași) să fie zero, făcând posibilă ieșirea din ciclu, altfel am avea un **ciclu infinit**.
6. **instrucțiune** poate fi orice instrucțiune recunoscută de C++, inclusiv instrucțiunea **do...while**. Când două sau mai multe instrucțiuni **do...while** sunt incluse una în alta, se spune că avem instrucțiuni **do...while imbricate**.

Exemple:

```
#include <iostream>
using namespace std;
int main()
{    // Program cu ciclu infinit. Iesire cu tastele Ctrl+Break
    do
    {

    }
    while(1);
    cout<<"DA";
    return 0;
}
```